Digital Objects as Passwords*

Mohammad Mannan and P.C. van Oorschot
Carleton University, Canada

Abstract

Security proponents heavily emphasize the importance of choosing a strong password (one with high entropy). Unfortunately, by design, most humans are apparently incapable of generating such passwords, or memorizing a random-looking, machine-generated one for long-term use. Infrequently used passwords pose even bigger security and usability problems. We exploit the fact that many users now own or have access to a large quantity of digitized personal or personally meaningful content in designing an object-based password scheme called `ObPwd`. ObPwd enables users to select a password generating object from their local collection or from the web, and then converts the password object (e.g. an image, a particular piece of music, excerpt from a book) to a (potentially) high-entropy text password that can be used for regular or secondary web authentication, or in local applications (e.g. encryption). Instead of requiring users to memorize an exact password, ObPwd only requires one to remember a hint or pointer to the password object used. We believe that choosing digital objects as passwords is an interesting alternative to explore, and may enable users to create and maintain high quality passwords. We have implemented a prototype, and solicit feedback from the research community in regard to using digital objects as passwords.

1 Introduction and Motivation

Despite all their shortcomings, text-based passwords are still heavily used by everyday users and security experts. Decades apart independent studies reveal that people consistently choose 'weak' passwords [14, 3, 21]. There are several apparent reasons for such behaviour. Strong or high-entropy passwords are difficult for users to generate, memorize, and reproduce at a later point in time. Also as the benefits of a strong password over a weak one are not readily noticed, there is little apparent motivation for users to spend extra effort in choosing strong passwords. Blaming users, or restricting password choice with complex rules (see e.g. [13, 22]) usually do not help. Alternatives to text-based passwords such as biometrics, hardware tokens, and two-factor methods are still far from being widely deployed, and password use is likely to dominate user authentication in the foreseeable future [6]. The existence and indeed recent rise in SSH password-guessing attacks[1] indicates that stronger passwords still increase security despite the proliferation of password stealing attacks, phishing and keylogging (cf. [4]).

Let us assume that with continual education, motivating efforts [5, 24], restrictions or proactive checking [27], users are persuaded to choose strong (random-looking, high entropy) passwords for everyday use. These strong passwords generally remain usable (i.e. well-remembered) only if used often. However, passwords employed to access rarely-used services, or in *secondary authentication* (e.g. when a user has lost/forgot the primary password) are not frequently recalled, motivating users to choose weak passwords/secrets that are difficult to forget or obvious when given a hint. We introduce an Object-based Password scheme called `ObPwd` which may be best used for passwords that are (i) infrequently used, or (ii) used for secondary or fall-back authentication, e.g., Password Verification Questions (PVQs); see for example, Rabkin [17] for a discussion of serious security weaknesses of PVQs as used in a number of current online banking sites.

The basic idea of ObPwd is the following. Many users currently possess a large collection of digital content such as photos, mp3s, and videos. Much of this content is *mobile*: users may keep it on personal devices (e.g. USB sticks, cellphones), or upload it to personal sites (in some cases, password-protected). Many users also have instant access to static content from the web, e.g., Internet Archive (`www.archive.org`), Project Gutenberg (`www.gutenberg.org`), and Google Books (`books.google.com`). An ObPwd password can be generated from such digital content as follows: compute a hash of user selected content, such as a photo file from the user's USB stick, and then con-

[*] Version: July 14, 2008. Email: `mmannan@scs.carleton.ca`. This paper appeared at USENIX HotSec, July 29, 2008.

[1] One experimental setup [18] reported an average of 2,805 SSH login attempts per computer per day.

vert the hashed bitstring to a password (a 'random-looking' string of keyboard characters or as an option, a human readable sequence of words using existing techniques [9, 12, 8, 19]). Users keep a record (memorized or written) of a pointer to their content used in generating each password. Users can write down the password in a 'secure' place, or re-create it from the content when needed. ObPwd requires no modifications to the software interface of password-based systems. Also, authenticating parties (remote or local) are not required to be aware of ObPwd (e.g. storing of a user's password-generating objects is not required).

ObPwd may offer the following benefits over existing techniques (see also Section 3).

1. Reduced Memory Load. Instead of requiring users to remember exact passwords or passphrases, ObPwd only expects them to recall a semantic *pointer* to their password object (e.g. hints for an image, video, entire/partial document, executable, URL, or highlighted text passage from a web page).
2. Resistance to Offline Dictionary Attack. Without having access to all of a user's possible password objects (from local media and web), attackers cannot build a password dictionary. Assuming password objects significantly vary among users (e.g. each user may have an independent collection of photos), creating a generalized password dictionary for ObPwd seems impractical. In contrast, building a dictionary of popular passphrases is apparently feasible [10], and general password dictionaries are already available [15].
3. Written Record of Passwords. In contrast to most graphical passwords [23], users can easily keep a written copy of ObPwd passwords (e.g. in a safe place as backup). Thus ObPwd enables converting an image-based password into human readable text (which also facilitates sharing – see below), and benefits from the easy memorability of object or image hints while keeping the simplicity of text passwords (easy deployment, written records).
4. Password Sharing. In cases where objects are already shared (e.g. photos, documents), ObPwd allows safer password sharing through a hint or description of the password object, without transmitting the actual password over the network. This seems preferable to some current practices such as sending shared passwords over email. It also allows sharing of the text-form output, which although often discouraged, may nonetheless be an important usability feature.

2 Object-based Password (ObPwd)

In this section, we discuss the ObPwd scheme in more detail, threat model, variants of the basic idea, and a prototype implementation.

Threat model, operational assumptions, and notation. We assume that password-generating objects in ObPwd are selected from a large public collection, e.g., files (including pdf) from the ACM digital archive (containing millions of archived documents), or from personal digital content (inaccessible to others). Hopefully either the large size of pools of such source objects, or the inaccessibility of private content will impede attempts to build offline dictionaries. Ideally users would not choose password objects from their (publicly accessible) personal website or public profiles as in Facebook/MySpace sites. (However, appending such objects with a salt apparently reduces some risks; see under 'Variants' below.) To enable *access-from-anywhere*, users either carry password-generating objects with them, or have online access to those objects. ObPwd passwords, and hints (text reminders) to password objects can optionally be written down. Passwords must be written down if a user does not want to carry content files with her.

If a password is directly generated from the password object and users 'copy-paste' that password instead of typing it in (see 'Implementation' below), keylogging attacks on passwords may be restricted. However, if ObPwd is used in regular web login, we strongly suggest that the password objects should be stored in local media (i.e. user devices) when passwords are generated on-the-fly (right before login). If a password is re-created from (plaintext) web content the following attack is possible. An attacker observes or records traffic from the intermediate network looking for a user to go into a content-hosting site right after or before requesting an authenticating website; thus the attacker can capture or narrow down candidates for the password-generating content. When ObPwd is used for encryption/decryption in a user's local media, getting access to password-generating objects from the network does not allow the attacker to gain any protected content (as the network attacker does not have access to the user's local encrypted files). Of course if the attacker already controls the user PC, neither ObPwd nor other password schemes can help. Similarly, this scheme is vulnerable to shoulder surfing and phishing (but see 'Variants' below). When a user has multiple password objects for different accounts/applications, the usual issue of password interference may also surface (which object is used for which account). However, ObPwd is focused to increase usability of a ‘strong’ password by leveraging distinctive object choices that might be made by a user, including leveraging their personal content. We use the following notation:

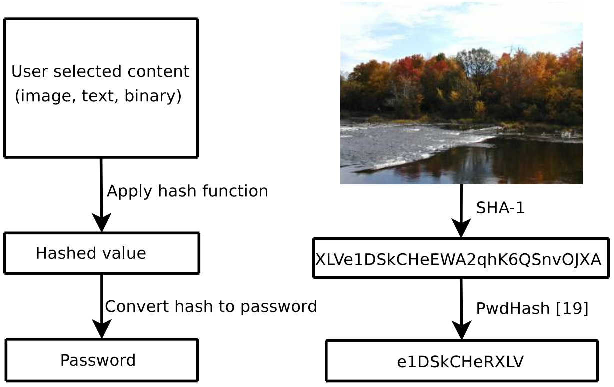
U	An ObPwd user.
M	A password object selected by U for a particular site or application.
$h(\cdot)$	An appropriate cryptographic hash function.
Hash2Text$(\cdot)$	A function (e.g. based on [9, 12, 8, 19]) for converting hashed bits into a string of keyboard characters, or optionally, words.
pwd	A password as generated by Hash2Text$(\cdot)$.

Steps in ObPwd. The steps in ObPwd are as follows; see also Fig. 1.

1. U selects an easy to remember object M from her personal media or from the web. To preclude offline dictionary attacks and predictable object prefixes, M should be required to exceed a minimum size (perhaps 30 bytes). Considering the time that may be required to hash very large objects (in step 2), such as a movie, M is ideally truncated to an appropriate number n of bytes (e.g. $n = 100000$).
2. U indicates the selected object to the ObPwd tool, which generates the hash H of M using a secure hash function h: $H = h(M)$.
3. H is used to generate $pwd = \text{Hash2Text}(H)$.

   H may be truncated depending on the required size of an output password. pwd (and M) should not be stored at the same place or media as the protected content. If used as a site password, pwd may require special encoding depending on the particular site; we do not address encoding issues separately here (but note that encoding techniques are addressed elsewhere [19]).

Variants. The basic idea of ObPwd can be extended as follows. A user-selected, ideally memorable *salt* string (s) may be appended as a second input to the hash function h: $H = h(M, s)$. The salt could be a 4-digit PIN, or a dictionary word. This enhancement may impede attackers even when a user's password object is exposed, albeit at the cost of memorizing a salt string. If used only rarely, then the salt need not be memorized but rather could be looked up from where it was written down. While the ObPwd scheme as proposed is vulnerable to phishing attacks, this weakness can be addressed, by appending the URL of a target site (as in [19], this can be done without user involvement) with the password object, i.e., $H = h(M, url)$.



(a) Generic steps in ObPwd (b) An example of ObPwd

Figure 1: ObPwd steps with an example

Implementation. We have implemented a basic prototype of ObPwd as a browser extension for Firefox (Fig. 2), and also as a stand-alone application in Windows XP (developed in `C#`). When a user clicks the right mouse button on a web object (an image, highlighted text, or a file URL), the browser extension inserts a menu item (e.g. ‘Get ObPwd from Image’ in Fig. 2) into the context menu; if selected, the extension generates a password from the underlying content and displays the password in a dialog box (Fig .3). In the local application, a user selects a particular file, which is then used as the password-generating object, and the password is displayed in a text box. For both implementations, we use SHA-1 as the hash function, and PwdHash [19] for converting hash values into a password (12 characters long, alphanumeric). We use at most $n = 100000$ bytes from a password object, and require a minimum of 30 bytes. Both implementations are available online (see Section 4). For mobility, if the ObPwd extension or application is not available from a remote computer, a website for generating passwords from user objects could be designed (cf. `pwdhash.com` [19]). We do not make any claims about the usability of the present prototype, but if the idea generates interest, would hope to pursue this and to host such a site.

3 Related Work and Comparison

There have been countless publications on passwords. Here we discuss only a selective subset of schemes designed to strengthen passwords (i.e. improving entropy) or to enhance usability (i.e. improving the ease-of-use). Infrequently used passwords such as Personal Verification Questions (PVQs) are discussed separately as ObPwd is apparently most suitable for these.

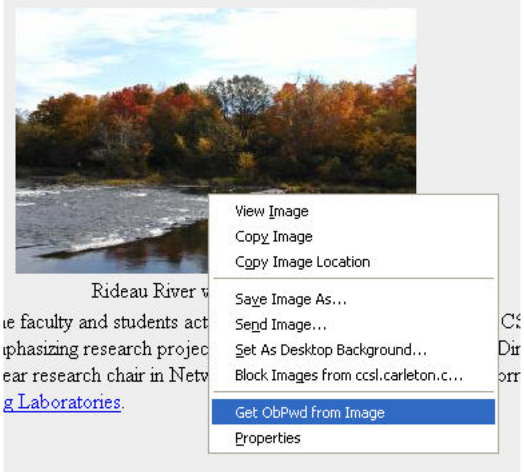
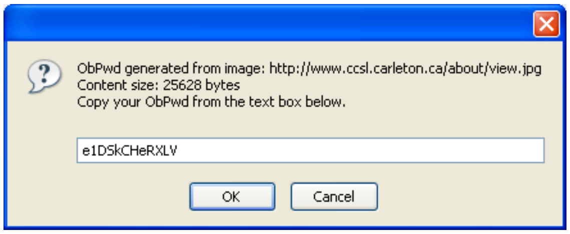

Rideau River w
ie faculty and students act
iphasizing research projec
ear research chair in Netw
g Laboratories.
CS
Dir
orr
View Image
Copy Image
Copy Image Location
Save Image As...
Send Image...
Set As Desktop Background...
Block Images from ccsl.carleton.c...
Get ObPwd from Image
Properties

Figure 2: ObPwd extension menu in Firefox


ObPwd generated from image: http://www.ccsl.carleton.ca/about/view.jpg
Content size: 25628 bytes
Copy your ObPwd from the text box below.

e1DSkCHeRXLV

Figure 3: Password generated from the selected image

3.1 Schemes for Improving Password Strength/Usability

Cheswick [1] proposed an obfuscated challenge-response based authentication scheme assuming people can *compute* a simple response to a given challenge according to a (user-selected) pass-algorithm. Both the challenge and response are obfuscated with *decoy* information. This scheme offers several desirable features (e.g. protection against keyloggers and phishing). Challenges noted by the author include users may forget the pass-algorithm/obfuscation technique more readily than a regular password, if used infrequently.

Florêncio et al. [4] argue that relatively weak passwords (e.g. with 20 bits of entropy) may provide enough security for web accounts assuming that: (i) a "three-strike" type rule (i.e. login is blocked after three failed attempts) is deployed to counter brute-force attacks; (ii) the user ID space is much larger than the IDs in actual use; and (iii) the valid user ID list is not readily available to attackers. Meeting these assumptions requires assistance from authenticating sites.

To improve password strength while maintaining usability, Forget et al. [5] proposed Persuasive Text Passwords (PTP) wherein system-generated characters are inserted at random positions into a user-chosen initial password. Users can accept the proposed password, or request (until satisfied) alternate suggestions. PTP essentially provides a middle ground between system-chosen (strong but difficult to remember) and user-chosen (weak but memorable) password schemes.

Yan et al. [26] conducted a user-study to compare regular user-chosen passwords, random passwords and mnemonic phrases. They reported finding that mnemonic phrases are as good as random passwords, and easier to remember. However, passphrases (and mnemonic passwords generated from them) may also be attacked by building a dictionary from commonly used phrases as available on the web [10].

Disk encryption software TrueCrypt allows users to use any local file along with a possibly empty password as an encryption key.[2] Users cannot write down the actual encryption key as a backup, and the generated key is used only with TrueCrypt. ObPwd was conceived independently.

Apparent advantages of ObPwd. In addition to web authentication, ObPwd passwords can arguably be used for applications which must withstand offline dictionary attacks (e.g. file encryption). Also, the deployment of ObPwd does not require any changes in system-side processing or password verification, or to the user interface in a web or local application.

ObPwd enables converting image-based passwords into text, and thus may be viewed as a middle ground between text and image-based password schemes. ObPwd can use (memorable) images while retaining simple advantages of text passwords (no-cost deployment, written records). While some people think writing passwords down and sharing passwords are poor practice, this arguably depends on the threat model, and usage. Certainly, being able to write down and backup infrequently used passwords seems essential. The fear of not writing down passwords may also encourage users to choose weak passwords.

Sharing of passwords (quite common in the real world; see e.g. [16]) in most graphical schemes is awkward if not impossible. ObPwd may enable better password sharing than text and graphical schemes without sacrificing confidentiality to third parties. For example, if two users share a digital photo folder (e.g. through personal media), then one user can choose a specific image as the password object, and send the other user a hint or description of the image (e.g. "our whitewater kayaking photo") over public media or email. Now an eavesdropper can see the hint,[3] but cannot generate the shared password without having access to the image object itself. Although this is in ef-

[2] This feature is apparently available since version 4.0 (Nov. 2005); see `http://www.truecrypt.org/docs/?s=keyfiles`.

[3] Here we assume that the hint is not an obvious link to a publicly-accessible web object.

fect equivalent to sharing a list of secret keys, arguably the advantage here is that we use more *meaningful* objects than randomly generated keys.

3.2 Personal Verification Questions

Personal Verification Questions (PVQs) are used for resetting a forgotten password or as part of login. While generally weaker than passwords, PVQ answers are typically equally useful to access an account. The availability of personal information on the web has apparently made it easier to correctly guess PVQ answers [17] (see also [7, 20]). As an example, Hollywood actress Paris Hilton's private photos and close contacts' phone numbers were exposed when an attacker was able to log into her T-mobile web account by answering her pet dog's well-known name to a PVQ [11].

Academic work on PVQ. Early work on PVQs includes *cognitive passwords*[4] and a related user study by Zviran and Haga [28]. It was reported that users could recall cognitive passwords more accurately than regular passwords. The authors also tested guessing attacks on cognitive passwords by significant-others of a user. Fact-based questions such as 'mother's maiden name' and 'name of your best friend in high school' were correctly guessed by 57% and 43% of users respectively. Opinion-based questions such as 'favourite colour' and 'last name of your favourite college instructor' were correctly guessed 41% and 10% of the time, respectively. The authors used 20 questions, of which users must answer five randomly selected questions at each login attempt.

Ellison et al. [2] proposed using personal questions and answers for recovering secret keys. Instead of using a passphrase, they require a user to pre-register n personal questions and answers (usually low-entropy), and then recover the secret key by correctly answering some $t < n$ of the questions. Thus a user can forget some answers, but still recover the secret.

Recently, Rabkin [17] analyzed over 200 PVQs as used in 17 financial websites. Taking the 'era of Facebook' into account, different classes of attacks are considered (e.g. random guessing, attacks automatically using online information, dedicated human attackers, and knowledge through personal acquaintance). As a possible defense, the following use of personal content was suggested. A user may upload an image of a person, and an answer to the question "what is the name of this individual?" However, as noted, any tagged photo of that person enables attackers to answer the question.

ObPwd as PVQs. If ObPwd is used in certain types of PVQ schemes (which allow free-format questions/answers), attackers cannot succeed without getting the actual password object. Many PVQ answers have quite low entropy ("What is your favourite colour?"). ObPwd password entropy is expected to be significantly higher. Also ObPwd requires no uploading of multimedia content to an authenticating site, and an exact copy of the password object is required for a successful attack (cf. [17]).

4 Concluding Remarks

Humans are not good at choosing high-entropy secrets that are easily memorable for a long time. Arguably, current password generation techniques and password-restricting rules have largely failed to improve password strength. Creating passwords from personally meaningful/memorable digital objects may be more user-friendly than any existing password rules; we emphasize, however, that we have not yet carried out any user testing. Depending on the application, variants of the basic ObPwd scheme may be suitable; for example, URLs can be appended to password objects (as in PwdHash [19]) if phishing is a concern.

Apparently passwords generated by our method would have more entropy than regular passwords. We have yet to devote serious attention to the question of determining defendable estimates of the security gains that might result, or a method to quantify guessability in the absence of very large-scale user trials (e.g. of millions of users). Indeed, despite existing password crack papers (e.g. [25]), it is not clear that the community even has a strong understanding of the empirical security of existing text passwords chosen by the mythical "typical user" for the mythical "typical password application". Studies of even as many as 500,000 users are too small for the long-tailed distribution of user-chosen passwords, and obtaining or publishing cleartext passwords in such studies is complicated by privacy concerns [3].

Our proposal has obvious limitations. Losing the pointer or the password object itself (if no written copy is kept) is equivalent to forgetting a regular password. Also, obvious and publicly accessible choices of password objects, e.g., the profile photo of a user's Facebook account, could result in even less security than text passwords. The potential security of ObPwd relies on the richness of the universe from which public objects are selected, and/or the inaccessibility of personal objects. ObPwd is introduced here to solicit feedback and promote discussion, to help advance the eternal quest for a better password scheme. We encourage readers to try out our implementation available at `http://www.ccsl.carleton.ca/~mmannan/obpwd/`.

[4] These are questions and answers related to a user's personal facts or opinions; they were designed to be used as regular passwords.

Acknowledgements. We thank anonymous HotSec 2008 reviewers for their comments and members of Carleton's Digital Security Group for enthusiastic discussion on this topic, especially David Barrera and David Whyte. The first author is supported in part by an NSERC CGS. The second author is Canada Research Chair in Network and Software Security, and is supported in part by an NSERC Discovery Grant, and the Canada Research Chairs Program.

References

[1] W. Cheswick. Johnny can obfuscate; beyond mothers maiden name. In *USENIX Workshop on Hot Topics in Security (HotSec'06)*, Vancouver, BC, Canada, July 2006.

[2] C. M. Ellison, C. Hall, R. Milbert, and B. Schneier. Protecting secret keys with personal entropy. *Future Generation Computer Systems*, 16(4), Feb. 2000.

[3] D. Florêncio and C. Herley. A large-scale study of web password habits. In *World Wide Web Conference (WWW2007)*, Banff, Alberta, Canada, May 2007.

[4] D. Florêncio, C. Herley, and B. Coskun. Do strong web passwords accomplish anything? In *USENIX Workshop on Hot Topics in Security (HotSec'07)*, Boston, MA, USA, Aug. 2007.

[5] A. Forget, S. Chiasson, P. van Oorschot, and R. Biddle. Improving text passwords through persuasion. In *Symposium on Usable Privacy and Security (SOUPS'08)*, Pittsburgh, PA, USA, July 2008.

[6] S. Furnell. Authenticating ourselves: will we ever escape the password? *Network Security*, 2005(3).

[7] V. Griffith and M. Jakobsson. Messin' with Texas, deriving mother's maiden names using public records. In *Applied Cryptography and Network Security (ACNS'05)*, New York, NY, USA, June 2005.

[8] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Trans. on Information and System Security (TISSEC)*, 2(3), Aug. 1999.

[9] N. Haller. The S/KEY one-time password system. In *Symposium on Network and Distributed System Security (NDSS)*, San Diego, CA, USA, Feb. 1994.

[10] C. Kuo, S. Romanosky, and L. F. Cranor. Human selection of mnemonic phrase-based passwords. In *Symposium On Usable Privacy and Security (SOUPS'06)*, Pittsburgh, PA, USA, July 2006.

[11] MacDevCenter.com. How Paris got hacked? News article (Feb. 22, 2005), `http://www.macdevcenter.com/pub/a/mac/2005/01/01/paris.html`.

[12] D. McDonald. A convention for human-readable 128-bit keys, Dec. 1994. RFC 1751 (Informational). `http://www.ietf.org/rfc/rfc1751.txt`.

[13] Microsoft. Strong passwords: How to create and use them. Online article (Mar. 22, 2006). `http://www.microsoft.com/protect/yourself/password/create.mspx`.

[14] R. Morris and K. Thompson. Password security: A case history. *Communications of the ACM*, 22(11), Nov. 1979.

[15] OpenWall.com. John the Ripper password cracker. `http://www.openwall.com/john/`.

[16] A. S. Patrick. Monitoring corporate password sharing using social network analysis. In *International Sunbelt Social Network Conference*, St. Pete Beach, Florida, USA, Jan. 2008.

[17] A. Rabkin. Personal knowledge questions for fallback authentication. In *Symp. on Usable Privacy and Security (SOUPS'08)*, Pittsburgh, PA, USA, July 2008.

[18] D. Ramsbrock, R. Berthier, and M. Cukier. Profiling attacker behavior following SSH compromises. In *IEEE/IFIP Dependable Systems and Networks (DSN'07)*, Edinburgh, UK, June 2007.

[19] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *USENIX Security Symposium*, Baltimore, MD, USA, 2005.

[20] B. Schneier. The curse of the secret question. Blog post (Feb. 11, 2005), `http://www.schneier.com/blog/archives/2005/02/the_curse_of_th.html`.

[21] B. Schneier. Real-world passwords. Analysis of 34,000 MySpace.com userid-password pairs. Blog post (Dec. 14, 2006). `http://www.schneier.com/blog/archives/2006/12/realworld_passw.html`.

[22] R. E. Smith. The strong password dilemma. Chapter 6 in "Authentication: From Passwords to Public Keys", Addison-Wesley, 2002. Excerpt available at `http://www.cryptosmith.com/sanity/pwdilemma.html`.

[23] X. Suo and Y. Zhu. Graphical passwords: A survey. In *Annual Computer Security Applications Conference (ACSAC'05)*, Tucson, AZ, USA, Dec. 2005.

[24] D. Weirich and M. A. Sasse. Pretty good persuasion: A first step towards effective password security in the real world. In *New Security Paradigms Workshop (NSPW)*, Cloudcroft, NM, USA, Sept. 2001.

[25] T. Wu. A real-world analysis of Kerberos password security. In *Network and Distributed System Security Symp. (NDSS'99)*, San Diego, CA, USA, Feb. 1999.

[26] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *IEEE Security & Privacy*, 2(5), Sept.-Oct. 2004.

[27] J. J. Yan. A note on proactive password checking. In *New Security Paradigms Workshop (NSPW)*, Cloudcroft, NM, USA, Sept. 2001.

[28] M. Zviran and W. J. Haga. Cognitive passwords: The key to easy access control. *Computers & Security*, 9(8), Dec. 1990.